**Lab Exercise 13– Snapshot Materialization in dbt Cloud for Snowflake**

Snapshot materialization in dbt allows you to capture changes in a dataset over time by creating historical records. This is useful for slowly changing dimensions (SCDs) or tracking changes in data.

In this lab exercise, we will:

1. Create a snapshot model in dbt.
2. Define the unique\_key and updated\_at columns for the snapshot.
3. Configure and execute the snapshot.

**Step-by-Step Lab Exercise:**

**Step 1: Set Up the Database and Source Table in Snowflake**

First, we need a source table in Snowflake that will be used for the snapshot. This table will contain customer data, and we'll track changes over time (e.g., if a customer's email address changes).

Create a customers table in Snowflake with sample data:

CREATE OR REPLACE TABLE raw.customers (

customer\_id INT,

name STRING,

email STRING,

address STRING,

created\_at TIMESTAMP,

updated\_at TIMESTAMP

);

-- Insert some sample data into the customers table

INSERT INTO raw.customers (customer\_id, name, email, address, created\_at, updated\_at) VALUES

(1, 'John Doe', 'john.doe@example.com', '123 Elm St', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP),

(2, 'Jane Smith', 'jane.smith@example.com', '456 Oak St', CURRENT\_TIMESTAMP, CURRENT\_TIMESTAMP);

**Step 2: Configure dbt Snapshot**

1. **Create a Snapshot Folder:**

In your dbt project, create a folder called snapshots/ if it doesn’t exist already. This is where we will define the snapshot model.

1. **Create the Snapshot Model:**

Create a new file called snapshots/customers\_snapshot.sql in the snapshots/ folder.

-- snapshots/customers\_snapshot.sql

{% snapshot customers\_snapshot %}

{{

config(

target\_schema='hks\_schema',

unique\_key='customer\_id',

strategy='timestamp',

updated\_at='updated\_at'

)

}}

SELECT

customer\_id,

name,

email,

address,

created\_at,

updated\_at

FROM {{ source('raw', 'customers') }}

{% endsnapshot %}

**Explanation:**

* + **unique\_key='customer\_id'**: This is the unique identifier for the record. dbt will track changes for each customer using this key.
  + **strategy='timestamp'**: This specifies that dbt should use the updated\_at column to track changes. This is common for Slowly Changing Dimension (SCD) type 2.
  + **updated\_at='updated\_at'**: This column is used to determine if a row has been modified. If the updated\_at value has changed since the last snapshot, dbt will insert a new version of the record.

**Step 3: Run the Snapshot**

Now, you can execute the snapshot to create and track historical records. Run the following dbt command to run the snapshot:

dbt snapshot --select customers\_snapshot

**Expected Behavior:**

* On the first run, dbt will take a snapshot of all records in the customers table and insert them into a target table with a name like customers\_snapshot in your defined schema.
* On subsequent runs, dbt will compare the updated\_at timestamp in the source table (raw.customers) and check for any changes. If any changes are detected (e.g., if a customer's email or address was updated), dbt will insert a new version of the record with the updated values and a new timestamp. The previous version of the record will be marked as "expired" (if using the timestamp strategy).

**Step 5: Verify the Snapshot in Snowflake**

After running the snapshot, you can verify the snapshot table in Snowflake by querying it. The table created for the snapshot will have a name like your\_schema.customers\_snapshot.

SELECT \* FROM your\_schema.customers\_snapshot;

**Expected Output:**

You should see records with the following structure:

| **customer\_id** | **name** | **email** | **address** | **created\_at** | **updated\_at** | **dbt\_valid\_from** | **dbt\_valid\_to** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | John Doe | john.doe@example.com | 123 Elm St | 2024-11-01 12:00:00 | 2024-11-01 12:00:00 | 2024-11-01 12:00:00 | NULL |
| 2 | Jane Smith | jane.smith@example.com | 456 Oak St | 2024-11-01 12:00:00 | 2024-11-01 12:00:00 | 2024-11-01 12:00:00 | NULL |

* **dbt\_valid\_from**: The timestamp when the record became valid.
* **dbt\_valid\_to**: If the record was updated, the previous version will have an end timestamp indicating when it was replaced by the new version.

**Step 6: Modify Data to Test the Snapshot**

Let’s simulate a change in the customers table by updating one of the records. For example:

UPDATE raw.customers

SET email = 'john.doe@newdomain.com', updated\_at = CURRENT\_TIMESTAMP

WHERE customer\_id = 1;

Now, run the snapshot again:

dbt snapshot --select customers\_snapshot

After running the snapshot again, you will see that the previous record for customer\_id = 1 will have a dbt\_valid\_to timestamp, and a new record will be added with the updated email.

**Summary**

In this lab, you learned how to:

* Set up a snapshot in dbt to track historical changes in data using the timestamp strategy.
* Configure a snapshot model with a unique\_key and updated\_at column.
* Use dbt’s snapshot functionality to track changes in your Snowflake database.

Snapshots are a powerful way to track changes in your data over time, especially for slowly changing dimensions (SCD type 2). By running snapshots periodically, you can build a history of data changes, which can be valuable for analytics, reporting, and auditing purposes.